SANDXXXX–XXXX
Unlimited Release
Printed December XXXX

PICO User Manual
Version 1.0

William E. Hart
Discrete Algorithms and Mathematics Department
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM
wehart@sandia.gov

Cynthia A. Phillips
Discrete Algorithms and Mathematics Department
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM
caphill@sandia.gov

Jonathan Eckstein
MSIS Department
Faculty of Management and RUTCOR
Rutgers University
640 Bartholomew Road
Piscataway, NJ 08854
jeckstei@rutcor.rutgers.edu

Abstract This documeft provides a user's guide for the PICO software library. PICO

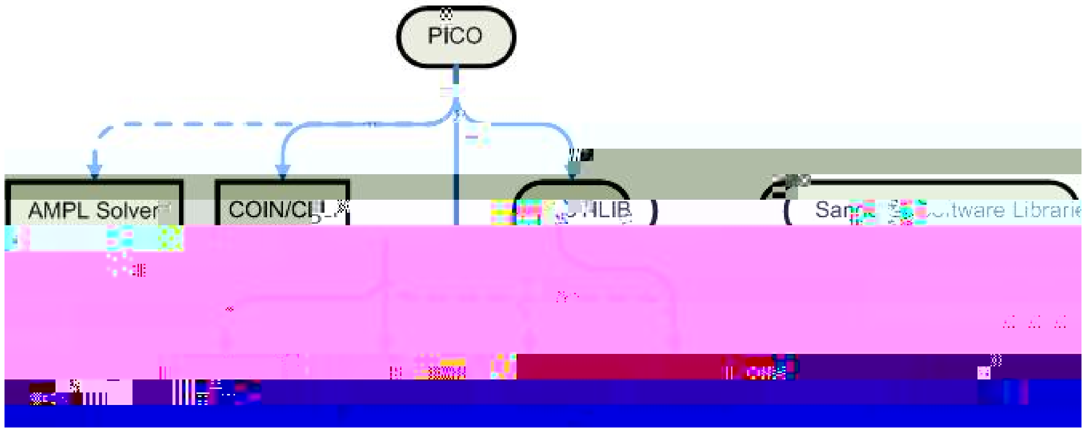

Figure 1: An illustration of inter-package dependencies within Acro that are used with PICO.

illustrates the dependencies between the Acro packages that are used with PICO. The dashed lines

cvs.a and ssh.cvs

These tools are available at http://software.sandia.gov.

The following mailing lists are used to manage Acro:

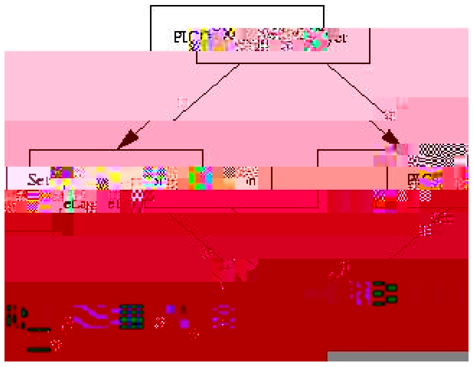

Figure 2: An illustration of how core PICO objects are used to derive a custom application solver.

PICO uses MPI because it is designed to be customized for maximum performance on MPP systems like the ASCI supercomputers. The design of PVM stresses the ability to operate on heterogeneous platforms, at some sacrifice in performance.

4.2 Extending the PICO Core

Defining a serial branch-and-bound algorithm with PICO requires the extension of two principal classes in the PICO serial layer: **branching** and **branchSub**. The **branching** class stores global information about a problem instance and contains methods that implement various kinds of serial

```
template <class FunctionT>
class parallelLipshitzian : public parallelBranching, public serialLipshitzian< FunctionT>
{
public:

  /// Return a new subproblem
  pico::parallelBranchSub* blankParallelSub();

  /// Pack the branching information into a buffer
  void pack(utilib::PackBuffer& outBuffer);

  /// Unpack the branching information from a buffer
```

```
int main(int argc, char* argv[])
{
try {
  /// Reset the UTILIB global timing information
  InitializeTiming();
  /// If we’re using MPI, then initialize the MPI data structures
  #if defined(USING_MPI)
  uMPI::init(&argc,&argv,MPI_COMM_WORLD);
  int nprocessors = uMPI::size;
  #else
  int nprocessors = 1;
  #endif
  FunctionClass problem;

  /// Do parallel optimization if we’re using more than one processor0-11.9d[(int)f (nprocessors > 1) {
     #if defined(USING_MPI)
     CommonIO::begin();
     CommonIO::setIOFlush(1);
     parallelLipshitzian<FunctionClass> optimizer;0-11.9d[(int)f (optimizer;0-Der;mlasr=
/// A simple quadratic problem
class FunctionClass
{
public:

 ///
 FunctionClass()
        {
        lower.resize(1l43o{
        lowes

        lowes


               ///
               rray<dopubl>&e
                      {

                      ;s[illegible]weo{

                      ///hle
```

```
 UP BOUND     C0002     1
 UP BOUND     C0003     1
ENDATA
```

If this file is named SC.mps, then we can directly apply PICO as follows:

```
    PICO SC.mod
```

References

[1] Q. Chen and M. C. Fearis. FATCOP: A fault toleaant Condor-PVM mixed integer progaamming solvea.